

# Dimorphic Computing: Sustainable Performance Through Thick and Thin

Andrés Lagar-Cavilla<sup>†</sup>, Niraj Tolia<sup>\*</sup>, Rajesh Balan<sup>\*</sup>, Eyal de Lara<sup>†</sup>, M. Satyanarayanan<sup>\*</sup>, David O’Hallaron<sup>\*</sup>

<sup>†</sup>University of Toronto, <sup>\*</sup>Carnegie Mellon University

## 1 Introduction

It is known that thick and thin clients have complementary strengths and weaknesses. Thick clients offer crisp interactive response even for tightly-coupled applications with rich GUIs. However, computationally intensive tasks are limited to using only the resources locally available. Thin clients, on the other hand, are attractive in CPU-intensive and data-intensive situations because application execution can occur on powerful remote compute servers or close to large data sets. Unfortunately, the physical separation between application execution and user interaction in thin clients can lead to poor application responsiveness, especially for tightly-coupled applications with rich GUIs.

As a result, neither thin nor thick clients are able to fully address the needs of a large class of applications that combine heavy computational demands with tightly-coupled interactive engines. Examples of these applications are found in the domains of digital animation, scientific computing, computer assisted design (CAD), or video editing. These applications typically interleave compute stages with interaction intensive stages. For example, an application might take many minutes to compute an optimized molecular model whose properties are then studied in an interactive visualization phase. A new model setting is then chosen and the sequence repeats itself.

We describe a new model of computing called *dimorphic computing*, that combines the strengths of thick and thin clients. During resource-intensive application phases, a dimorphic client behaves like a thin client. During interaction-intensive phases, it behaves as a thick client. Transitions are completely transparent and seamless to the user, who just sees excellent performance at all times: *short completion time* for computationally-intensive phases, and *crisp response* for interaction-intensive phases.

We introduce an open-source tool called *AgentISR* that realizes the model of dimorphic computing. It leverages Virtual Machine (VM) migration technology to move the execution site of an application, and complements it with a specialized mechanism for VM disk state transfer and an automated policy for triggering migrations. While there have been many previous approaches to computation mobility, *AgentISR* presents four key improvements. First, applications do not have to be modified, recompiled, or relinked. This greatly simplifies real-world deployments where use of proprietary applications may be unavoidable. Second, the application does not have to be written in a specific language, nor does it need to be built using specific libraries. By requiring almost nothing of applications except the existence of distinct computation and interaction phases, *AgentISR* in-

vites the broadest possible usage. Third, our approach is especially clean in its separation of policy and mechanism. The code to decide when to migrate an application is completely independent from the code that implements the relocation. Finally, our approach is transparent to the end user.

## 2 Design and Implementation

*AgentISR* uses the Xen Virtual Machine Monitor (VMM) [2] to implement the concept of an *agent*, a migratable embodiment of an application<sup>1</sup>. *AgentISR* isolates an unmodified application in its own VM and exploits virtual machine migration techniques to dynamically relocate the agent across physical hosts, thus switching between thick and thin client computing. Because applications execute on top of the OS for which they were developed, *AgentISR* can run any application binary without requiring modifications. VM migration is achieved by suspending a running VM and transferring the image to another host where it is resumed, or by iteratively copying the live VM’s memory in a process known as live-migration [3]. Live migration has the advantage of only suspending the VM for a few seconds, rendering the transition effectively transparent to the user.

While standard migration techniques focus on transferring the VM’s memory image, the virtual disk of an agent (typically several GBs in size) needs to be efficiently migrated as well. For this, *AgentISR* uses a custom distributed block device called *WANDisk*. *WANDisk* allows for the availability of persistent caches of the VM disk at the destination hosts, and the efficient transfer of the deltas between caches upon VM migration. A VM disk cache is partitioned into chunks of 128 KB, and a chunk table is used to keep track of versioning and ownership information. The chunk table is used to identify stale chunk misses and trigger the appropriate updates on-demand.

To decide when to migrate an agent, *AgentISR* uses an automated migration policy. The policy gathers and processes VM resource consumption data from a set of sensors; standard sensors include CPU utilization, network usage, and interactivity measurements. The sensor readings trigger transitions in a finite state machine. Each state of the machine represents an application’s specific need and an associated location: by migrating the agent to that location the best performance is achieved. Agent state machines are defined by application-specific profiles; profile generation does not involve any application modifications, and can be done by users, third-parties, or even the developers themselves. However, we envision the deployment of machine learning algorithms to automate the process of decision making.

<sup>1</sup>Other VMMs like VMWare that provide similar functionality could also be used.

### 3 Validation Approach

In this section we discuss initial results of our AgentISR evaluation. We are currently experimenting with four different applications that are used for video editing, 3D animation, quantum chemistry, and earthquake modeling. In the interests of space, we only present the results from *QuakeViz*, an interactive visualization of a 500MB output file produced by a simulation of an earthquake in an  $80 \times 80 \times 12 \text{ km}^3$  region of the Los Angeles basin [1]. In a processing intensive *crunch phase*, *QuakeViz* extracts ground motion isosurfaces from this dataset, and applies several transformations to generate a visually appealing result. An ensuing interactive *visualization phase* lets users examine the isosurfaces by zooming, rotating, and panning.

To evaluate AgentISR’s performance, we conducted a set of controlled experiments using traces of long interactive user sessions. The purpose of these experiments was to measure both crunch and visualization phase performance. We employed a session replay mechanism to replay traces of previously recorded sessions under three execution models:

**Thick Client:** Runs the application on the user’s desktop, which connects to *QuakeViz*’s dataset via a WAN link.

**Thin Client:** Executes the application on a machine that is connected to the dataset via a Gigabit link.

**AgentISR:** Migrates the application between the above two machines, with an artificially optimal migration policy.

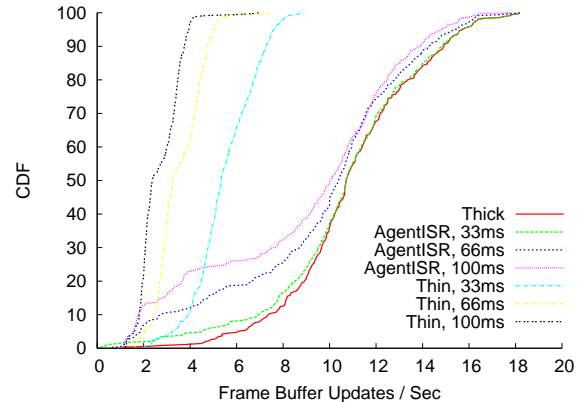
For the above three models, the bandwidth on the WAN link was set at 20 Mbit/s and the latency was set to either 33, 66, or 100 ms. Figure 1 shows the total time to completion results. The results shows the clear disadvantages of executing the crunch phase at the user’s desktop. *QuakeViz* needs to fetch a large dataset through the WAN link, impacting time to completion by as much as 13 minutes. Both the AgentISR and thin client cases fare significantly better, as they execute the crunch phase at a location closer to the dataset.

Network RTT	Thick	Thin	AgentISR
33 ms	51.8 (0.0)	46.8 (0.1)	47.9 (0.2)
66 ms	57.4 (0.2)	46.7 (0.1)	47.0 (0.1)
100 ms	63.4 (0.6)	46.7 (0.0)	47.4 (0.1)

Each data point is the mean of three trials; standard deviations are in parentheses. The visualization phase took approximately 21 minutes for all experiments.

**Figure 1:** Total Time to Completion

Figure 2 shows the cumulative distribution of frame buffer update rates, obtained by measuring the frames per second yielded by the response of each input during the visualization phase (mainly mouse button presses and releases while rotating or zooming an isosurface.) AgentISR offers interactive performance similar to thick client execution, and dramatically better than thin client computing. Even in the worst case of 100 ms RTT, approximately 70% of the event responses offer smooth frame rates, closely matching those obtained with thick client execution. Thin client computing



**Figure 2:** CDF of Frame Buffer Updates Rates

Network RTT	Total Migration Time	Suspend Time
33 ms	165.0 (0.7)	2.9 (0.3)
66 ms	323.5 (1.1)	6.6 (1.2)
100 ms	436.3 (1.5)	6.9 (1.9)

These results show the time (in seconds) taken to complete live migration for *QuakeViz*. Each data point is the mean of three trials; standard deviations are in parentheses.

**Figure 3:** Migration Performance

is, by nature, unable to provide the high frame rates that result from the ability to leverage local computing resources. In particular, most interactive responses yield at most half the frame rates than in the thick client or AgentISR cases.

The disparity in frame rates for the AgentISR cases is a direct result of relocation time. Figure 3 shows that for higher network RTTs, more time is spent in Thin Client mode during live-migration. The suspend column indicates the time for the last iteration of live-migration, for which the agent is unresponsive: in all cases it is a negligible figure. It is well-known that the effects of increased RTT can be factored out through TCP buffer-size adjustment, effectively reducing all migration times (and frame rates curves) to the 33ms case.

In conclusion, AgentISR is a system that enables dimorphic computing through the automated use of virtual machine migration techniques. AgentISR achieves good performance for compute-intensive stages and provides a crisp interactive experience during intensely interactive phases. These benefits are achieved even for demanding network conditions and in a manner transparent to end-users and developers.

### References

- [1] AKCELIK, V., BIELAK, J., BIROS, G., EPANOMERITAKIS, I., FERNANDEZ, A., GHATTAS, O., KIM, E. J., LOPEZ, J., O’HALLARON, D., TU, T., AND URBANIC, J. High resolution forward and inverse earthquake modeling on terascale computers. In *Proc. ACM/IEEE Supercomputing* (Phoenix, AZ, Nov. 2003).
- [2] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)* (Bolton Landing, NY, Oct. 2003), ACM Press, pp. 164–177.
- [3] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In *Proc. 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (Boston, MA, May 2005).